2 A Dialogue with Professor Joseph Sifakis about Concurrent Systems Specification and Verification

Professor Joseph Sifakis, `http://www-verimag.imag.fr/~sifakis`*, is a leading researcher well-known for his pioneering work in theoretical and practical aspects of concurrent systems specification and verification, notably the area of model-checking. His current research activities include component-based design, modelling, and analysis of real-time systems with focus on correct-by-construction techniques.*

Educated at the Electrical Engineering at the National Technical University of Athens and the University of Grenoble, Professor Sifakis received a doctorate (1974) and a state doctorate (1979) from the University of Grenoble, and a Dr. h.c. from the École Polytechnique Fédérale de Lausane, Switzerland (2009).

Professor Sifakis holds the INRIA-Schneider endowed industrial chair and works at the Verimag Laboratory, Grenoble, of which he is a founder. He is the director of the Carnot Institute "Intelligent Software and Systems" in Grenoble (`http://www.carnot-lsi.com`*) and coordinates Artist2, the European Network of Excellence for research on Embedded Systems.*

Professor Sifakis has received the CNRS Silver Medal in 2001 and the Turing Award for 2007. He is a grand officer of France's national order of merit.

Cristian Calude: Please tell us about your education and the people who influenced most your academic career.

Joseph Sifakis: I studied Electrical Engineering at the National Technical University in Greece. As a student I was inclined to be more concerned with theory than with practice. I came to Grenoble in 1970 for graduate studies in Physics. An encounter has been decisive for my career: I had the chance to meet Professor Jean Kuntzmann, who was the Director of the Institute of Informatics and Applied Mathematics (IMAG). My interest in Computer Science grew and I decided to quit my studies in Physics and start undergraduate studies at IMAG. Jean Kuntzmann was an inspired mathematician. I did my Engineering thesis under his supervision on modelling the timed behaviour of circuits. I learned a lot from him, and I remember that he always recommended not looking at the bibliography before coming up with one's own solution to a problem. This is a rule that I have strictly observed throughout my career. After my Engineering Thesis, I became interested in the theory of concurrency.

In 1974, I met Carl Adam Petri and then visited him and his colleagues in Bonn several times. I was really impressed by his erudition but I could not really

understand why "true concurrency" was such a big idea. In contrast to the prevailing approach, I considered in my papers that Petri nets are merely transition systems.

In 1977, I definitely left Petri nets for program semantics and verification. Dijkstra's papers and books had a deep influence on my work as well as discussions with Michel Sintzoff who was working at that time on program verification. They drew me the idea of fixpoint characterization for temporal modalities, and this opened the way to the results on model-checking. My student Jean-Pierre Queille developed the first model checker in 1982. I met Ed Clarke and Allen Emerson at CMU in November 1982 and we realized that we had been working independently on the same problem. From Patrick Cousot I learned about abstract interpretation—our offices were in the same corridor and we were both members of the "Programming Languages" team.

In the autumn of 1983, I met Amir Pnueli for the first time at a workshop on "The Analysis of Concurrent Systems", organized in Cambridge. This was the beginning of a continuous interaction and collaboration for more than 25 years. Amir had a deep influence on my career. For more than 10 years, we have setup several European projects in collaboration with Willem-Paul de Roever, on system modelling and verification. We jointly organized with Ed Clarke the Workshop on the "Verification of Finite State Systems" in Grenoble in 1989. This workshop is considered as the first edition of the CAV Conference. Amir Pnueli opened my horizons and contributed to the visibility and recognition of our work at Verimag through his international network of connections and collaborations. He brought me into contact with leading researchers and teams working on timed and hybrid systems. Thomas Henzinger came to Verimag as a postdoctoral student. Oded Maler, Amir Pnueli's Ph.D. student, joined my team as a permanent researcher. We had a strong team that contributed significantly to the development of the state-of-the-art in hybrid and timed systems. Amir Pnueli has frequently visited Verimag for over ten years and we greatly benefited from his wisdom and support.

The interaction and collaboration with researchers in my own team also had a deep impact on my career. I will mention Ahmed Bouajjani, Saddek Bensalem, Susan Graf, Oded Maler, Sergio Yovine and Stavros Tripakis as well as Paul Caspi and Nicolas Halbwachs who designed and developed the Lustre synchronous programming language.

In the late 90's my research interests progressively shifted from verification and formal methods to system design. This was not the result of the direct influence of a single person but rather through an increasing awareness that verification was hitting a wall and only incremental improvements in the state-of-the-art could be expected. I stepped down from the Steering Committee of CAV and started a research program on embedded systems design. Interactions with colleagues such as Hermann Kopetz, Lothar Thiele, Thomas Henzinger, Alberto Sangiovanni-

Vincentelli and Edward Lee contributed to elaborating a system perspective for Computer Science. I worked actively for setting up the Emsoft Conference and for organizing the Embedded Systems community in Europe through the Artist coordination measure followed by the Artist2 and ArtistDesign European Networks of Excellence. Although this incurred some considerable effort in administrative work, I learned a lot through the scientific management of world class research teams.

CC: Please present Verimag you have founded and directed.

JS: In January 1993, I founded Verimag laboratory, a joint-venture between IMAG (Computer Science and Applied Mathematics Institute) and Verilog SA. This has been an exiting and fruitful experience. Verimag has transferred the Lustre language to the SCADE synchronous programming environment. SCADE has been used by Airbus for over 15 years to develop safety critical systems and has become a de facto standard for aeronautics. SCADE has been qualified as a development tool by the FAA, EASA, and Transport Canada under DO-178B up to Level A. It is currently been commercialized by Esterel Technologies. We also transferred functional testing and verification techniques to the ObjectGeode tool for modelling real-time distributed applications. This tool has been commercialized by Telelogic purchased by IBM in 2008.

Since 1997, Verimag has been a public research laboratory, associated with CNRS and the University of Grenoble. It plays a prominent role in embedded systems by producing cutting edge research and leading research initiatives and projects in Europe.

As the director of Verimag I have sought a balance between basic and applied research. I have used resources from industrial contracts and collaborative projects to develop new research activities and strengthen the potential in basic research. For me, participation in industrial projects has been a source of inspiration. It allowed the definition of new research directions that are scientifically challenging and technically relevant. I believe that this virtuous cycle of interaction between academic research and applications is the key to Verimag's success.

CC: What are Artist2 and ArtistDesign?

JS: ArtistDesign is a European Network of Excellence federating the European research community in Embedded Systems Design. It brings together 31 of the best research teams as core partners, 15 Industrial and SME affiliated Industrial partners, 25 affiliated Academic partners, and 5 affiliated International Collaboration partners who participate actively in the technical meetings and events.

The central objective for ArtistDesign is to build on existing structures and links forged in the FP6 Artist2 Network of Excellence, to become a virtual Center of Excellence in Embedded Systems Design. This is mainly achieved through

tight integration between the central players of the European research community. These teams have already established a long-term vision for embedded systems in Europe, which advances the emergence of Embedded Systems as a mature discipline.

The research effort aims to integrate topics, teams, and competencies, through an ambitious and coherent research programme of research activities which are grouped into 4 Thematic Clusters: "Modelling and Validation", "Software Synthesis, Code Generation, and Timing Analysis", "Operating Systems and Networks", "Platforms and MPSoC", "Transversal Integration" covering both industrial applications and design issues aims for integration between clusters.

The NoE has a very dynamic International Collaboration programme, interacting at top levels with the best research centers and industrial partners in the USA: (NSF, NASA, SRI, Boeing, Honeywell, Windriver, Carnegie Mellon, Vanderbilt, Berkeley, UPenn, UNC Chapel Hill, UIUC, etc) and in Asia (Tsinghua University, Chinese Academy of Sciences, Seoul National University, East China Normal University, etc).

ArtistDesign also has a very strong tradition of Summer Schools, Graduate Courses, and major workshops.

CC: Why is model checking so important for today's IT industry? What are your main contributions in this area?

JS: The first results on "property verification by evaluation of formulas" are in my Thesis (Thèse d'Etat) presented in June 1979. These results have been published in the paper "A Unified Approach for Studying the Properties of Transition Systems"—Theoretical Computer Science, Vol. 18, 1992. They include a fixpoint characterization of a simple logic with two modalities: possible and inevitable.

The results of my Thesis led to the development of the first model checker CESAR, in 1982. The tool allows translation of finite state CSP programs into Petri nets, extended with finite-valued variables. The verification method is symbolic, representing sets of model states as boolean expressions. My team developed in the 80's several model-checkers for the verification of distributed systems, by using enumerative techniques such as CADP and the IF toolbox. We also developed in collaboration with Telelogic the TGV testing tool that generates test suites for communication protocols from their specification in a simple temporal logic.

My research for more than a decade focused on increasing the efficiency of model-checking techniques. I investigated compatibility between equivalences induced by temporal logics and behavioural equivalences based on bisimulation relations that can be used to reduce models. I also produced results relating model-checking and abstract interpretation that have been successfully applied in tools such as Invest at Verimag and SAL at SRI.

In the early 90's we studied, in collaboration with T. Henzinger, X. Nicollin

and S. Yovine, the first symbolic model-checking algorithm for the verification of TCTL. This algorithm has been implemented in the Kronos tool at Verimag. I also worked on the verification of hybrid systems, in collaboration with researchers from Verimag and Rajeev Alur and Thomas Henzinger. These general results are complemented by work on the verification of specific classes of hybrid systems, in particular in collaboration with Amir Pnueli. Finally, in collaboration with Oded Maler and Amir Pnueli, I produced results on the synthesis of controllers for timed systems. These have been applied for schedulability analysis of real-time systems.

Today model-checking is a mature technology used by companies such as Intel, IBM and Microsoft. These have developed proprietary technology for verifying complex systems. Model-checking can be also used for debugging or generating suites for testing real implementations. I see two main obstacles to the application of model checking to complex systems. One is of course the size of the state space which may increase exponentially with the number of the components of a system. The other, equally important obstacle is generating faithful models from system description formalisms, in particular for mixed software/hardware systems.

CC: What is component-based construction and BIP?

JS: We need theory, models and tools for cost-effectively building complex systems by assembling heterogeneous components. This is essential for any engineering discipline. It confers numerous advantages such as productivity and correctness.

System designers deal with heterogeneous components, with different characteristics, from a large variety of viewpoints, each highlighting different dimensions of a system. They often use several semantically unrelated formalisms e.g. for programming, HW description and simulation. This breaks the continuity of the design flow and jeopardizes its coherency. System development is often decoupled from validation and evaluation.

System descriptions used along a design flow should be based on a single semantic model to maintain its overall coherency by guaranteeing that a description at step n+1 meets essential properties of a description at step n. The semantic model should be expressive enough to express different types of heterogeneity:

* Heterogeneity of computation: The semantic model should describe both synchronous and asynchronous computation to allow in particular, modeling mixed hardware/software systems.

* Heterogeneity of interaction: The semantic model should enable the natural and direct description of various mechanisms used to coordinate the execution of components including semaphores, rendezvous, broadcast, method call, etc.

* Heterogeneity of abstraction: The semantic model should support the description of a system at different abstraction levels from application software to

its implementation.

Existing theoretical frameworks for composition are based on a single operator e.g., product of automata, function call. Poor expressiveness of these frameworks may lead to complicated designs: achieving a given coordination between components often requires additional components to manage their interaction. For instance, if the composition is by strong synchronization (rendezvous), modeling the broadcast requires components for choosing the maximal amongst several possible strong synchronizations. We need frameworks providing families of composition operators for the natural and direct description of coordination mechanisms such as protocols, schedulers and buses. These should provide a unified composition paradigm for describing and analyzing the coordination between components in terms of tangible, well-founded and organized concepts. In addition, they should be equipped with tractable methods for ensuring correctness-by-construction to avoid the limitations of monolithic verification. These methods use two types of rules:

* Compositionality rules for inferring global properties of composite components from the properties of constituent components e.g. the composition of deadlock-free components is - under some conditions - a deadlock-free component. A special and very useful case of compositionality is when a behavioural equivalence relation between components is a congruence. In that case, substituting a component in a system model by a behaviourally equivalent component leads to an equivalent model. Today, we lack compositionality theory for progress properties as well as platform-dependent properties.

* Composability rules ensuring that essential properties of components are preserved when they are used to build composite components. Consider for instance, two components. One is the composition of a set of components sharing a common resource accessed in mutual exclusion. The other is obtained as the composition of the same set of components scheduled for optimal use of the shared resource. Is it possible to obtain a single component integrating this set of components and such that both mutual exclusion and the scheduling constraints hold? System engineers face this type of non-trivial problem every day. They use libraries of solutions to specific problems and they need methods for combining them without jeopardizing their essential properties. Feature interaction in telecommunication systems, interference among web services, interference in aspect programming are all manifestations of the lack of composability.

This vision has motivated my research over the past decade, during which I studied BIP (Behaviour, Interaction, Priority), a component framework for rigorous system design. BIP allows the construction of composite hierarchically-structured components from atomic components characterized by their behaviour and their interface. Components are composed by the layered application of interactions and of priorities. Interactions express synchronization constraints between

actions of the composed components while priorities are used to filter amongst possible interactions and to steer system evolution to meet performance requirements e.g. to express scheduling policies. Interactions are described in BIP as the combination of two types of protocols: a) rendez-vous to express strong symmetric synchronization and b) broadcast to express triggered asymmetric synchronization. BIP offers a clean and abstract concept of architecture separated from behaviour. Architecture in BIP is a first class concept that can be analyzed and transformed. BIP relies on rigorous operational semantics that have been implemented in three Execution Engines for centralized, distributed and real-time execution. The combination of interactions and priorities to describe coordination between components confers BIP expressiveness not matched by any other existing formalism. The usual notion of expressiveness does not take into account features such as primitives for structuring and composition. It considers as equivalent (Turing complete) a wide variety of formalisms from high-level programming languages to counter machines. I have proposed a notion of expressiveness that characterizes the ability of modeling formalisms to describe coordination mechanisms between components.

CC: What is your vision for the development of computer science?

JS: Computer Science is a young and rapidly evolving discipline due to the exponential progress of technology and applications. It is a scientific discipline in its own right with its own concepts and paradigms. It deals with problems related to the representation, transformation and transmission of Information. As such, it studies all aspects of computing from models of computation to the design of software and computing devices.

Information is an entity distinct from matter and energy. It is a resource that can be stored, transformed, transmitted and consumed. It is immaterial but needs media for its representation. Information is any structure to which one can assign a meaning. The number “4” can be represented by the symbols “100”, “four”, “IV”. All these representations have the same meaning defined by a semantic function. This concept is different from physical information measured as entropy in Information Theory and Physics, which characterizes the informational content of a specific representation.

Computer Science is not merely a branch of Mathematics. As any scientific discipline, it seeks validation of its theories on mathematical grounds. But mainly and most importantly, it develops specific theory intended to explain and predict properties of computations which can be tested experimentally.

More than 95% of the chips produced today are for embedded applications. These are electronic components integrating software and hardware jointly and specifically designed to provide given functionalities, which are often critical. They are hidden in devices, appliances and equipment of any kind: mobile phones,

cameras, home appliances, cars, aircraft, trains, medical devices etc. In 2008, a person used about 230 embedded chips every day: 80 chips in home appliances, 40 chips at work, 70 chips cars, 40 chips in portable devices. In the near future, another anticipated important landmark will be the advent of the Internet of Things as the result of a convergence between embedded technologies and the Internet. The idea is to use internet technologies to integrate services provided by hundreds of billions of embedded systems. This will require an upgrade of the internet infrastructure to make it more secure, safe and reactive. Current features for exchanging multimedia documents will be extended to encompass real-time monitoring and control. Systems are becoming ubiquitous. The state of almost everything can be measured, sensed and monitored. People and objects can communicate and interact with each other in entirely new ways. Intelligent systems allow enhanced predictability of events and optimal use of resources.

It is hard to imagine what Computer Science will be in two decades. More than any other discipline, it is driven by applications and exponential progress in technology. The broadening of its perimeter is accompanied by a shift in focus from programs to systems.

CC: Please illustrate with a simple example the difference between programs, on one hand, and systems, on the other hand.

JS: Programs usually compute a single function. They transform input data into output data. They must terminate and are deterministic. A system interacts continuously with a physical environment. It combines the execution of several functions. Its behaviour can be understood as a relation between input data streams and output data streams. It is in general non-terminating and non-deterministic. Consider for instance, a controller for a lift. Depending on external stimuli (pushing a button by a user) it will execute a function that moves the cabin to a destination. Its behaviour is non-terminating and may be non-deterministic.

Existing models of computation deal with functions. They ignore physical time and resources. Computation is a finite sequence of steps corresponding to the execution of primitive operations. Complexity theory is based on abstract notions of resources such as time and memory. Programs have behaviour that is independent from the physical resources needed for their execution. In contrast, essential systems properties strongly depend on physical resources.

New trends in computing systems bring Computer Science closer to Physics. Marrying physicality and computation requires a better understanding of their differences and points of contact. Is it possible to define models of computation involving quantities such as physical time, physical memory and energy? There exist significant differences in approaches and paradigms adopted by the two disciplines. We badly need holistic rigorous design approaches taking into account the interaction of mixed software/hardware systems with their physical environ-

ment.

CC: What and how should we teach CS?

JS: Computer Science complements and enriches our knowledge with theory and models enabling a deeper understanding of discrete dynamic systems. It proposes a constructive and operational view of the world which complements the classic declarative approach adopted by Physics.

Computer Science curricula seldom recognize the importance of systems and fails to provide a holistic view of the discipline. I have the following recommendations.

* Teach students how to think in terms of systems (design process, tools, interaction with users and physical environment). Computer Science curricula should be extended and enriched by including principles, paradigms, techniques from Control Theory and Electrical Engineering.

* Teach principles rather than facts (foundations, architectures, protocols, compilers, simulation...). Very often courses are descriptive and present details that can be acquired later as needed in professional life. Students should be prepared to deal with the constant change induced by technology and applications. They also should be kept aware of the limitations of existing theory of computing. Very often theory makes assumptions oversimplifying reality.

* Put emphasis on information and computation as universal concepts applicable not only to computers and provide the background for triggering critical thinking, understanding and mastering the digital world.

CC: Theories are not famous for leading to technologies. What do you think about research in Computer Science and its impact on industrial practice?

JS: Unfortunately, the current scope and focus of research in Computer Science fail to address central problems for the IT industry, in particular problems raised by system design and engineering. Following well-beaten paths rather than taking the risk of exploring new ideas is a prevalent attitude by researchers from all scientific communities. More than in other disciplines, research in Computer Science has been over-optimistic regarding the possibility for solving hard problems and overcoming obstacles. This can probably be explained by the strong demand and incentives for innovation by funding agencies as well as the strong push from applications and market needs. Very often, scientific roadmaps and position papers present "challenges" that are mere visions and take desires for reality. All of the following were once hyped as main breakthroughs: Artificial Intelligence, Fifth Generation Computers, Program Synthesis, True Concurrency, Web Science. The proper goal of theory in any field is to make models that accurately describe real systems. Models can be used to explain phenomena and predict system behaviour. They should help system builders do their jobs better. A very common

attitude is to work on mathematically clean theoretical frameworks whether or not they are relevant. Very often, simple mathematical frameworks attract the most brilliant researchers who produce sterile "low-level theory", that has no point of contact with real computing. This leads to a separation between theoretical and practical work that is harmful for the discipline. The opposite extreme is also observed. There exist frameworks intended to describe real systems such as UML and AADL constructed in an ad hoc manner. These include a large number of semantically unrelated constructs and primitives. It is practically impossible to obtain rigorous formalizations and build any useful theory for such frameworks. We need theoretical frameworks expressive enough to directly encompass a minimal set of high-level concepts and primitives for system description and amenable to formalization and analysis.

CC: Is it possible to find a mathematically elegant and still practicable theoretical framework for computing systems?

JS: Computer Science deals with building artefacts. The key issue is constructivity, that is, the ability to effectively build correct systems. As system synthesis from requirements is intractable for complex systems, we should study principles for building correct systems from components. The aim is to avoid a posteriori monolithic verification as much as possible. There already exists a large body of constructivity results in Computer Science such as algorithms, architectures, protocols. Their application allows correctness for (almost) free. How can the global properties of a composite system be effectively inferred from the properties of its constituents? This remains an old open problem that urgently needs answers. Failure in bringing satisfactory solutions will be a limiting factor to system integration. It would also mean that Computer Science is definitely relegated to second class status with respect to other disciplines.
Useful Links:
Joseph Sifakis' home page: `http://www-verimag.imag.fr/~sifakis`,
Verimag's home page: `http://www-verimag.imag.fr`,
ArtistDesign: `http://www.artist-embedded.org/artist`,
BIP: `http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html?lang=en`

CC: Thank you.